Performance Analysis of Different Optimization Methods for Linear Regression using Batch/Stochastic Computation and Data Resampling Strategies.
FYS-STK3155 - Project 1

Alexander Umansky
Norwegian University of Science and Technology, University of Oslo
(Dated: September 28, 2025)

In this study, we use different methods for polynomial interpolation of Runge function. We compared standard approaches using Linear Regression and Gradient Descend methods. Performance was evaluated based onMean squared error, stability of model coefficients, and convergence rate. Applying regularization proved: tbd. Compared to linear regression, batch gradient descend with constant learning rate did tbd. Using adaptive learning rate improved tbd. Using stochastic GD did tbf. Using mini-batching with imporance sampling showed tbd. We estimate optimal model complexity using bias variance tradeoff. We select small data sample and use Bootstrap and Kfold to reproduce it from the same distribution. For linear methods, maximal polynomial degree should not exceed $P = 10$. Prediction variance can be attenuated using larger data samples or introducing regularization.

I. INTRODUCTION

The optimization problem is central to any field of research, where we seek to match observations with data generated by models. Intuitively, the closer the alignment between the true and synthetic data, the more likely the model is to correctly represent reality. However, how should "matching" be measured, and wether good alignment always signifies correctness of the model, are central topics of Machine Learning and this study in particular.

This study is limited to Linear Regression i.e. the features are polynomial degrees x^n up to some maximal model complexity $n \leq P$ and the optimization problem is to find polynomial coefficients $\hat{\beta}$ such that $y \simeq \hat{\beta}X$. Though we can assume that the parameters are independent of the input parameters, models of high complexity can be computationally demanding, hence we investigate performance of matrix and Gradient Descend (GD) based approaches. Performance will be based on the mean squared error (MSE), stability and consistency of coefficient values, and for GD methods, also the convergence. We will investigate how adding regularization λ to Ordinary Least Square (OLS) method affects the performance, and how updating the learning rate η increases the convergence rate of GD methods.

The final topic of the analysis concerns how limited datasets can be used more effectively to improve optimization and training. In the context of Gradient Descend we introduce Mini-Batching and Stochastic Gradient Descend (SDG) with Normal sampling and possible extension to importance sampling. We also discuss Bootstrapping and K-fold cross-validation as two resampling methods to regenerate test data. We study the balance between model complexity and amount of training data required for optimal prediction- bias and variance.

II. THEORY AND METHOD

A. The Runge Function

In this work we study the Runge function $y(x) = \frac{1}{1+25x^2}$ on the interval $x \in (-1, 1)$ polluted by normally distributed noise $N(0, \sigma \leq 1)$. The choice of the Runge function is not a coincidence as it is a case where polynomial interpolation of increasing order does not converge and instead produces larger oscillations near the edges of the interval [*]. An analogy exits in Fourier analysis, called the Gibbs phenomenon, where approximating non-smooth functions with higher modes produces oscillations that never truly die out. Limited in model complexity, one solution involves using orthogonal series, such as Chebyshev polynomials, to control the oscillations [*]. In this project, we stay true to polynomial regression, in order to stress test the different methods.

B. Preprocessing and Evaluation

Once the $(N \times P)$ feature matrix X is constructed, where N is number of data points and P is model complexity (e.g. highest polynomial degree), we standardize each feature column to have zero mean and unit variance. This is done to ensure that each feature initially has equal weighting in the analysis. Another reason is that ill-conditioned feature matrices will accumulate floating-point errors during repeated matrix operations [*]. In later sections where we introduce regularization, feature scaling will become even more important in context of penalization.

The data is split into training and test subsets using a 1:4 ratio. Model parameters are optimized for the training data, while the test data is used to evaluate how good the model is at generalization by predicting unseen data. Before training, we scale using a 'standard scaler' provided by scikit-learn library to scale X_{test} and X_{train}.

We subtract the mean value from y_{test}, which will be added later during evaluation as an offset. This prevents scaling from intruding bias into the optimization.

One performance metric will be the test error. Here we define the Mean Squared Error and R^2:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \tag{1}$$

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{2}$$

where $\tilde{y} = X_{test}\hat{\beta}$, and $\bar{y}$ is the mean of test data y. The attractiveness of MSE is first of all due to its simplicity, and how it seamlessly emerges in so many different domains like Ordinary Least Squares in linear algebra or Bias-Variance in statistician analysis, thereby bringing them into the context of optimization.

The R^2 metric normalizes MSE by dividing it by the variance of y. Hence R2 is less dependent on preprocessing and its values are dimensionless.

C. Optimization

We obtain the optimal parameters by minimising the cost function, without regularization λ defined as ordinary least squares (OLS) or the MSE from before:

$$\mathcal{C}_{OLS}(\beta) = \frac{1}{n}\|X\boldsymbol{\beta} - y\|_2^2 = \frac{1}{n}(y - X\beta)^T(y - X\beta) \tag{3}$$

The optimal parameters are obtained by minimizing the cost function i.e. $\nabla\mathcal{C}_{OLS}(\beta) = 0$. If X and y are non-stochastic and independent of parameters, an algebraic solution for β exists:

$$\hat{\beta}_{OLS} = (X^TX)^{-1}X^Ty \tag{4}$$

Here we define the Hessian matrix $H = X^TX$. The Hessian happens to be the second derivative of the cost function and proportional to the covariance matrix. Its diagonal elements are variances, while its eigenvalues describe the type of optimization problem[1].

The matrix solution for the optimization problem relies on the existence of a matrix inverse of H which may well not be the case for high complexity and large degree of dependencies between the feature columns. The solution is to use Singular Value Decomposition (f.ex. NumPy's pseudoinverse), or introduce a small regularization λ to the diagonal terms in H. We define the Ridge and Lasso regularization methods:

$$\begin{aligned}\mathcal{C}_{\text{Ridge}}(\beta) &= \frac{1}{n}\|X\beta - y\|_2^2 + \lambda\|\beta\|_2^2 \\ \implies \hat{\beta}_{\text{Ridge}} &= (X^TX + \lambda\mathbf{I})^{-1}X^Ty\end{aligned} \tag{5}$$

$$\mathcal{C}_{Lasso}(\beta) = \frac{1}{n}\|X\beta - y\|_2^2 + \lambda\|\beta\|_1 \tag{6}$$

The second term in equations 6 and 5 is called the penalty term. Regularization parameter does not just solve the issue of invertability, it realizes a size constraint on the parameter coefficients:

$$\hat{\beta}_{\text{Ridge}} = \arg\min_{\beta}\sum_{i=0}^{N}-1\left(y_i - \beta_0 - \sum_{j=1}^{P-1}x_{ij}\beta_j\right)^2 \tag{7}$$
$$\text{subject to } \sum_{j=1}^{P-1}\beta_j^2 \leq t.$$

This expression simply rewrites and merges the L2 norms of Eq 5, so the t is related to λ [1]. Written this way, one can see that neglecting feature scaling results in uneven penalty between jth features. If before, scaling X with some diagonal matrix D ($D^T = D$) simply scaled the coefficients $\hat{\beta} = (DDX^TX)^{-}DX^Ty$, with regularization this is no longer the case. Notice also $y_i - \beta_0$. By subtracting the intercept $\simeq \bar{y}$ we make the penalty independent of data offsets [1].

When model parameters are highly correlated (such is the case for linear interpolation), regularization is a way to enforce stability of coefficients, the second performance milestone. Finally, the fact that Lasso method has a L1 penalty term, means the solution is no longer linear in y as the gradient will contain a term $sgn(\beta)$. Hence, no algebraic solution exists for the Lasso regularization unlike Ridge or OLS.

D. Statistical Interpretation

Consider $y = f(x) + \varepsilon$ where ε is some noise parameter given by $N(\mu = 0, \sigma)$ and f is assumed non-stochastic, independent on distributions of y and X. Then, the expected values of the model prediction $\tilde{y} = X\hat{\beta} \simeq f(x)$ and the true value y are both given by $X\hat{\beta}$. If we want to express the expected value of their squared differences as in Eq. 1 it can be shown [2] that $\mathbb{E}[(y - \tilde{y})^2]$ relates MSE to bias and variance of the prediction sample i.e. the distance from the average prediction to the true value, and the dispersion of predictions.

$$\mathbb{E}\big[(y - \tilde{y})^2\big] = \underbrace{\mathbb{E}\big[(y - \mathbb{E}[\tilde{y}])^2\big]}_{\text{Bias}} + \underbrace{\mathbb{E}(\tilde{y} - \mathbb{E}[\tilde{y}])^2\big]}_{\text{Variance}} + \underbrace{\sigma^2}_{\varepsilon\text{ noise}} \tag{8}$$

[1] If the Hessian is semi positive definite, meaning its eigenvalues are $\lambda_H \geq 0$, this implies a convex optimization problem i.e. finding the local minima of the cost function $\mathcal{C}(\beta)$.

The benefit of this will become clear when we analyze what model complexity to stop at. For instance, a polynomial of high degree can be fitted to one set of data and have a seemingly good MSE value, yet fluctuate away from the true function in some other region. This model will give good results for a particular test data, but exhibit high prediction variance.

E. Gradient Descend

In this section we introduce Gradient Descend (GD) as a more rigid optimization method. For high model complexities, computing the inverse (with or without SVD) becomes computationally unaffordable. Besides, any optimization problem which is dependent on parameters (f.ex. Lasso) has no analytical expression to take advantage of. GD builds upon the Newton method to find the roots of the cost function iteratively. We can expect fastest convergence when following the negative gradient of the function i.e. $\nabla\mathcal{C}(\beta)|_{\beta^n} \equiv g^{(n)}$ such that:

$$\beta^{(n+1)} = \beta^{(n)} - \eta g^{(n)} \tag{9}$$

where η is the Learning Rate. For cases where analytic solution still exists, we essentially traded inversion of an inner product of two $N \times P$ matrices $\mathcal{O}(NP^2 + P^3)$, for less demanding iterative computation of the gradient $\mathcal{O}(NP)$ per iteration. Of course, the Gradient Descend has its own shortcomings, for once, re-computation of the gradient can too become inefficient. Besides, the major downside of GDs is that the methods finds only *local* minima and is very dependable on initial conditions.

One practical solution is to subdivide test data into M mini-batches of some size S, and let each approach independently their local minima. The full data set (batch mode) and Stochastic Gradient Descend (batches consist of individual points) are limit cases of Mini-Batching. Each iteration we randomly construct M batches E times where E stands for epoch. The global update for one iteration becomes:

$$\beta^{(n+1)} = \beta^{(n)} - \eta \sum_{e}^{E} \sum_{m}^{M} \sum_{i \in B_m}^{S} \nabla_\beta \mathcal{C}_i(x_i, \beta) \tag{10}$$

Another downside is the gradient is "isotropic" in parameter-space. We could include higher order expansion, involving the Hessian, but this is again a computational issue. Instead we opt to "modify" the learning rate η. If set to small or too high GD will fail to converge efficiently. Hence, the eta must reflect the smoothness of the direction i.e. be a function of the gradient history. In this report we consider adaptive learning rate methods: Momentum, AdaGrad, RMSProp and ADAM.

1. Momentum

Using the standard approach behind Verlet algorithms one can introduce memory by considering previous and current iterations when computing future.

$$\begin{aligned} \beta^{(n+1)} &= \beta^{(n)} - \eta g^{(n)} + \delta\big[\beta^{(n)} - \beta^{(n-1)}\big] \Leftrightarrow \\ v^{(n)} &= \beta v^{(n-1)} + (1-\beta) g^{(n)} \\ \beta^{(n+1)} &= \beta^{(n)} - \eta v^{(n)} \end{aligned} \tag{11}$$

2. Adaptive Gradient (AdaGrad)

Adaptive Gradient method accumulates gradient squared, and the value to shirinks the learning rate

$$\begin{aligned} r_j^{(n)} &= r_j^{(n-1)} + g_j^{(n)} g_j^{(n)} \\ \beta_j^{(n+1)} &= \beta_j^{(n)} - \frac{\eta}{\sqrt{r_{n,j}} + \epsilon} g_{n,j} \end{aligned} \tag{12}$$

here ϵ prevents division by zero, and double indexing $g_j^{(n)}$ denotes accumulative value of the gradients.

3. RMS Propagation (RMSProp)

If the learning rate diminishes to quickly with Adagrad, copy the step averaging v_t from momentum based approach and use it with squared gradient:

$$\begin{aligned} r^{(n)} &= \rho r^{(n-1)} + (1-\rho)(g^{(n)})^2 \\ \beta^{(n+1)} &= \beta^{(n)} - \frac{\eta}{\sqrt{r_n} + \epsilon} g^{(n)} \end{aligned} \tag{13}$$

4. Adaptive Momentum (ADAM)

Adam method upholds both the momentum and the rms moving averages:

$$\begin{aligned} r_1^{(n)} &= \rho_1 r_1^{(n-1)} + (1-\rho_1) g^{(n)}, \quad \hat{r}_1^{(n)} = \frac{r_1^{(n)}}{\sqrt{1 - n\rho_1}} \\ r_2^{(n)} &= \rho_2 r_2^{(n-1)} + (1-\rho_2)(g^{(n)})^2, \quad \hat{r}_2^{(n)} = \frac{r_2^{(n)}}{\sqrt{1 - n\rho_2}} \\ \beta^{(n+1)} &= \beta^{(n)} - \eta \frac{\hat{r_1}^{(n)}}{\sqrt{\hat{r}_2^{(n)}} + \epsilon} \end{aligned} \tag{14}$$

where we used $n\rho_i$ instead of $\rho_i^{(n)}$.

F. Bootstrap and K-fold Cross Validation

Resampling techniques are used to regenerate new test data, which is particularly important when the dataset is sparse. In bootstrapping, the training sample is repeatedly resampled with replacement. This allows duplicate values to occur, but preserves the distribution of X_{test}. K-Fold Cross validation partitions the X sample into K "folds". For K iterations, a new fold serves as test data while the other are used for training. The predictions are recalculated K times, and then averaged.

In this report, we apply resampling to evaluate model complexity by examining how bias, variance and MSE change as function of P and N. Since N must remain relatively small to deduce optimal P,we generate additional data using $B > N$ bootstraps. We run a larger analysis, comparing OLS to Ridge, using K-Fold cross validation with $K \in (5, 20)$.

G. Resources

For the simulations, we wanted to rely as much as possible on our own code, ensuring full control over the methods, scaling and resampling techniques. The same preprocessing scheme was used across all comparisons, so that any observed differences can be attributed solely to the parameter we changed on purpose.

Notable library usage:

- **Scikit-learn**: StandardScaler, `train_test_split`, KFold and resample.
- **NumPy**: linalg.norm, pseudoinverse, lin. algebra
- **Matplotlib**: plot generation.

We also made active use of OpenAI ChatGPT and DeepSeek to identify bugs and logical inconsistencies in both the code and our comparative approach.

The code with necessary packages can be accessed here: https://github.com/4Lexium/Data-Analysis-and-Machine-Learning/tree/main/project1

III. RESULTS AND DISCUSSION

A. Preprocessing and Ordinary Least Squares

In section II A we did not specify the variance of the noise component. We have tried different parametrization of noise and found $\sigma = 0.3$ to be harsh enough for our trials. Figure 1 demonstrates the noisy distribution with $\sigma = 0.3$. The yellow line highlights the true bell-shaped slope. The largest concern is overfitting near the edges. There, the model will use higher polynomials to fit points that are displaced due to noise. Since the error evaluation using MSE uses test data with the same problem, it will not detect this as an issue. Using the target function would be "a posteriori", undesirable for generalization.

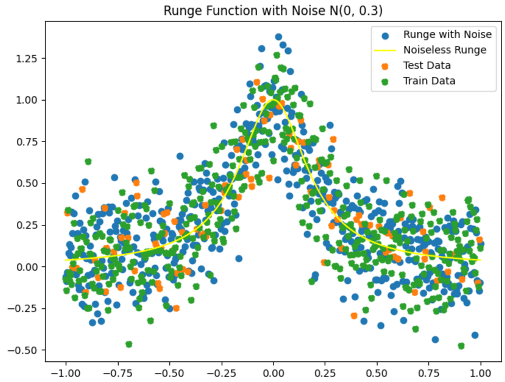

Figure 1: The preprocessed noisy ($\sigma = 0.3$) data sample (blue) is split into train and test subsets (green and orange) and plotted along with the noiseless Runge function (yellow). The y_{offset} is intentionally left out during optimization and added back during prediction evaluation.

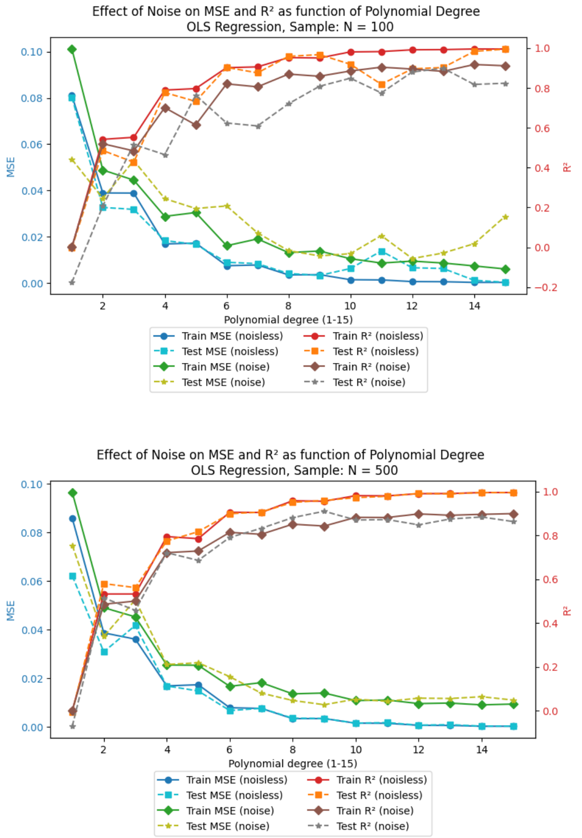

Figure 2: Error estimates using MSE and R2 are used here to compare performance of OLS method on a noiseless sample and one polluted by N($\mu = 0, \sigma = 0.1$). Despite the evident worsening when introducing noise, when large samples are used $N = 500$ the error fluctuations are controlled.

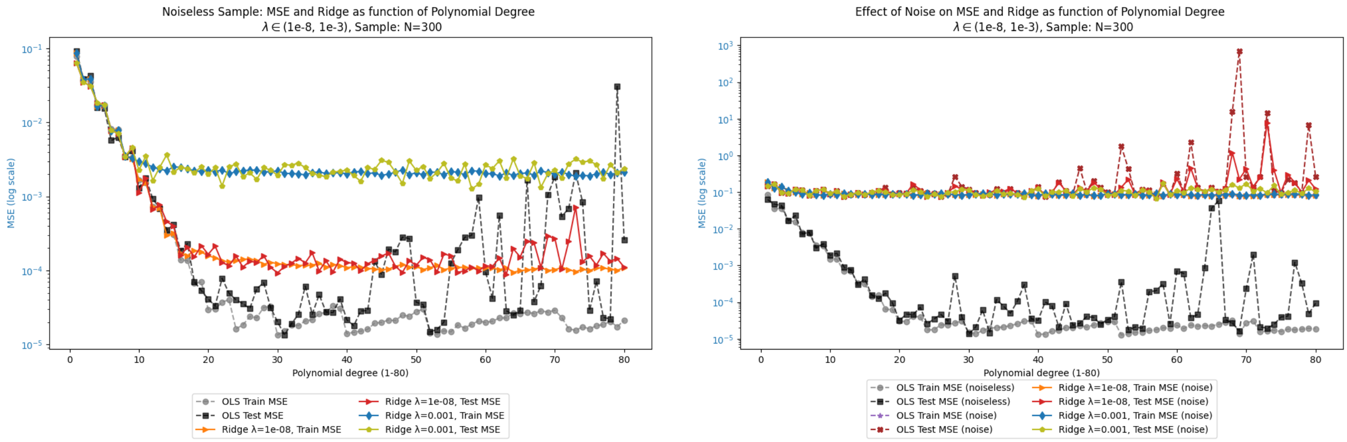

Figure 3: Error estimates using MSE and R2 are used here to compare performance of OLS method on a noiseless sample and one polluted by N($\mu = 0, \sigma = 0.1$).

Figure 7 shows results of OLS optimization of a data sample with noise $\sigma = 0.1$. The data was preprocessed (section II B) beforehand, so the scale dependent MSE and scale independent R2 agree on performance. The former decreasing to 0 while the latter approaching 1 when as more features are added. Notice already after $P = 10$, the improvement starts to stagnate and what began as random jumps in prediction error, resembles more divergence.

At this stage we use a mild noise level and relatively normal complexity. This test shows in particular how sample size affects OLS. If enough points are used in training, even if globally, noisy prediction still worsens, the error fluctuations are more controlled. There is also no sign of divergence at this concrete complexity interval. This topic will be re-visited in section III C.

B. Introducing Regularization

(Heatplots) Not sure why test is more noisy than train...Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

(OLS vs Ridge)Here we notice ... Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

C. Bias-Variance and Model Complexity

We use MSE and prediction Bias and Variance of a small data sample to test how the model generalizes. The sudden steepening in variance signifies overfitting, a notification that the model complexity is too high. On the other hand, low variance and high bias implies the model is still too simple. We seek a balance, hence the name tradeoff which usually lies in the minimum of the resulting curve. Using bootstrap method to resample the small test data we conclude that the optimal complexity for really small data should be kept around $P = 6$. To include higher polynomials, consider using larger natural test samples or regularization. From Fig. **??** we remember that regularization can not be too high. A moderate value (we tested with $\lambda = 10^{-5}, 10^{-3}$) prolonged the overfitting and makes it more gradual. This is linked to penalizing the coefficients.

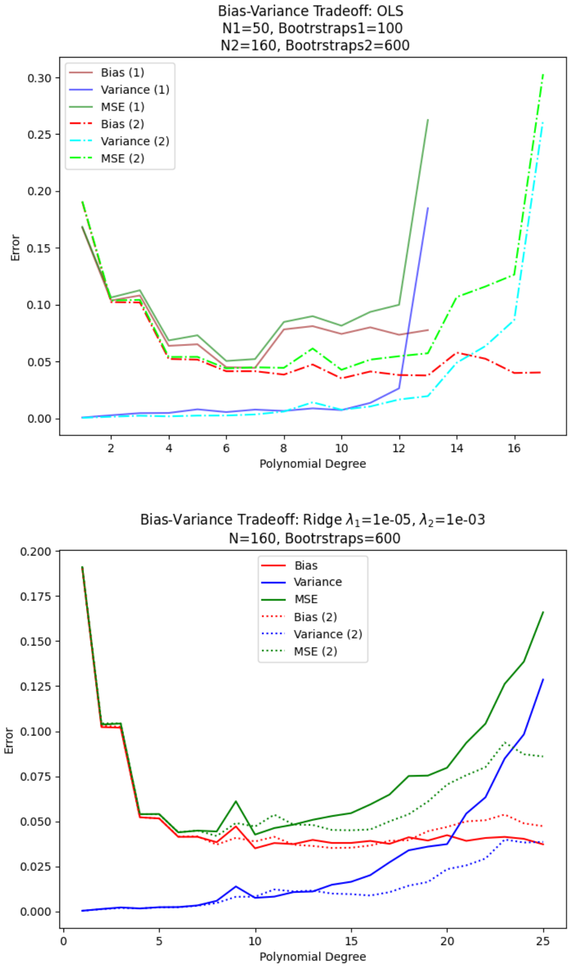

Figure 4: Bias-Variance tradeoff for two different train samples with different number of bootstraps. The minimum of each MSE curve determines the optimal complexity. Using larger set of points extends the optimal degree before the prediction variance starts to grow signifying overfitting. We draw the same conclusion that the optimal complexity is under $P = 10$. Higher degree without drastic overfitting can be achieved with regularization or using more sample points as was demonstrated in **??**.

We used Bootstrap to plot bias and variance as function of complexity for specific sample sizes see **??**, **??**. For method comparison we use Kfolding with $K = 10$, to avoid having to tune number of bootsraps inside the analysis for P see **??**.

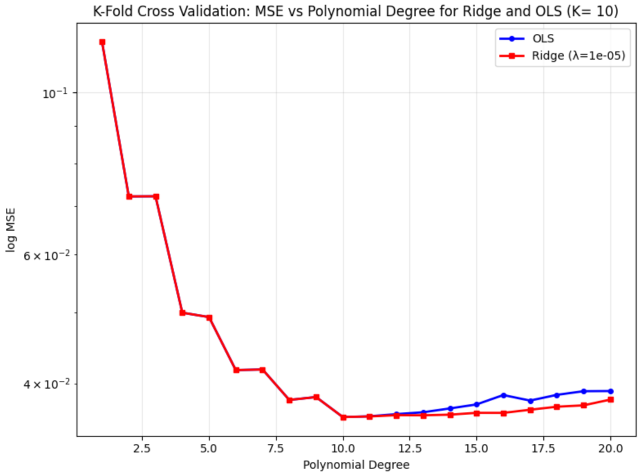

Figure 5: Performance of $K = 10$ folds on a dataset of 200 data points. We observe the same optimal complexity for both OLS and Ridge. For high degrees the MSE is still well contained with regularization since the prediction variance increases more gradually with regularization applied.

D. Gradient Descend Methods with Adaptive learning rate

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

IV. LARGE FIGURES

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

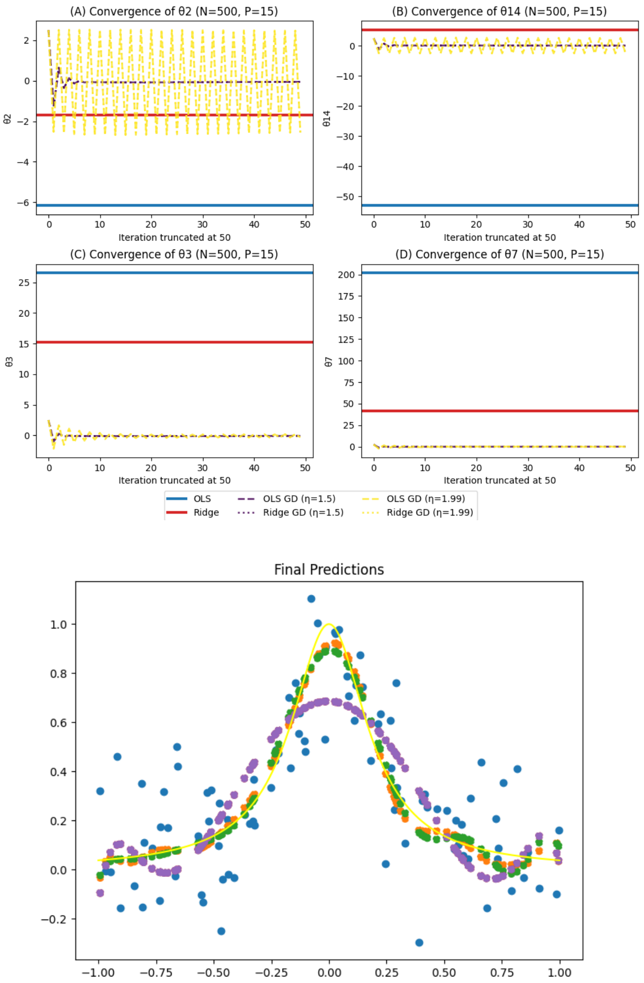

Figure 6: Simple GD

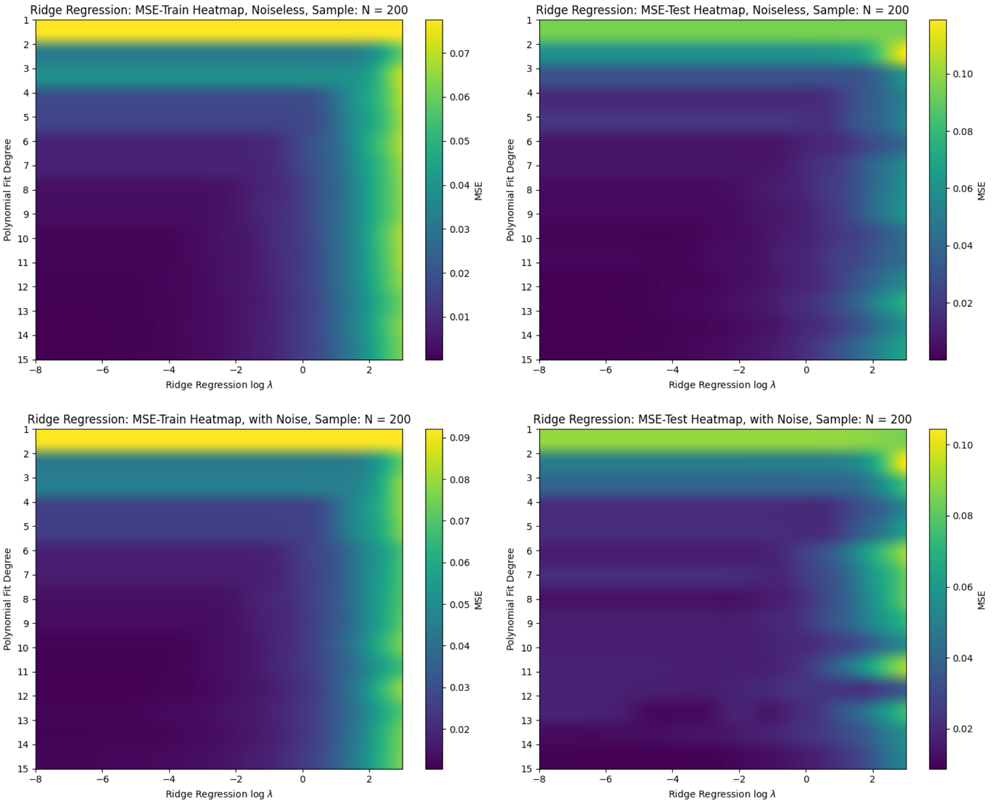

Figure 7: The Heatmap shows MSE of Ridge regression as a function of the regularization parameter λ and model complexity P. The MSE gradient is only drastic when λ is chosen very high. In this case we optimize with respect to the penalty term and not the feature matrix itself, effectively fitting with a constant vector. Aside from this edge case the ridge parameters are relatively stable.

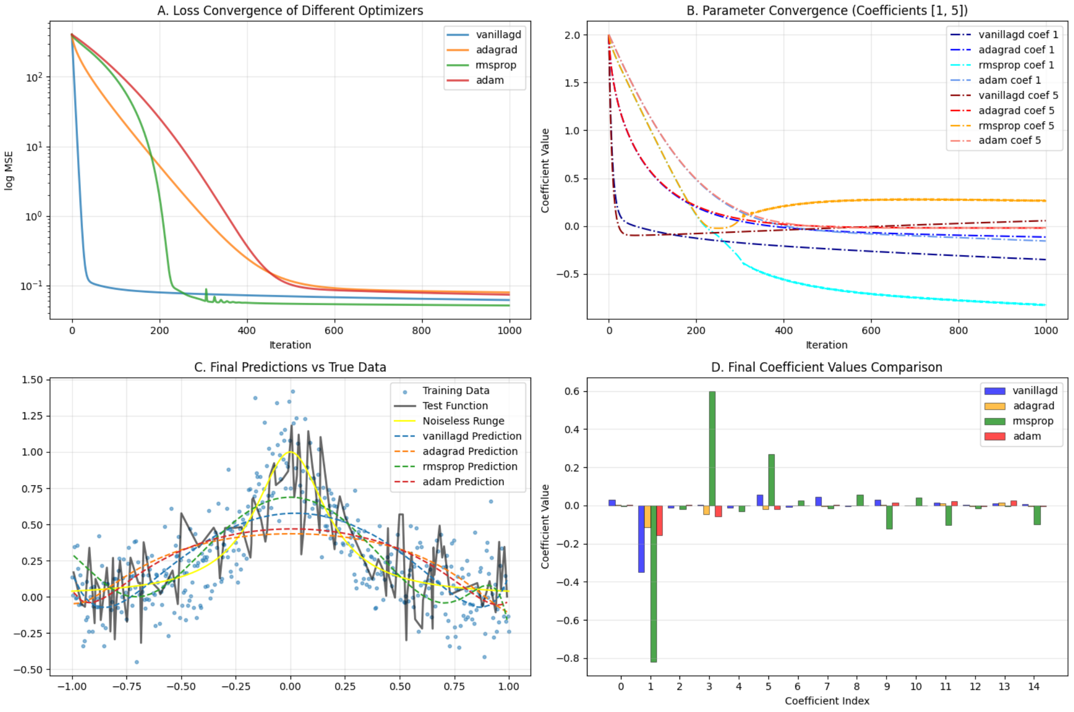

Figure 8: Advanced Learning Rate GD

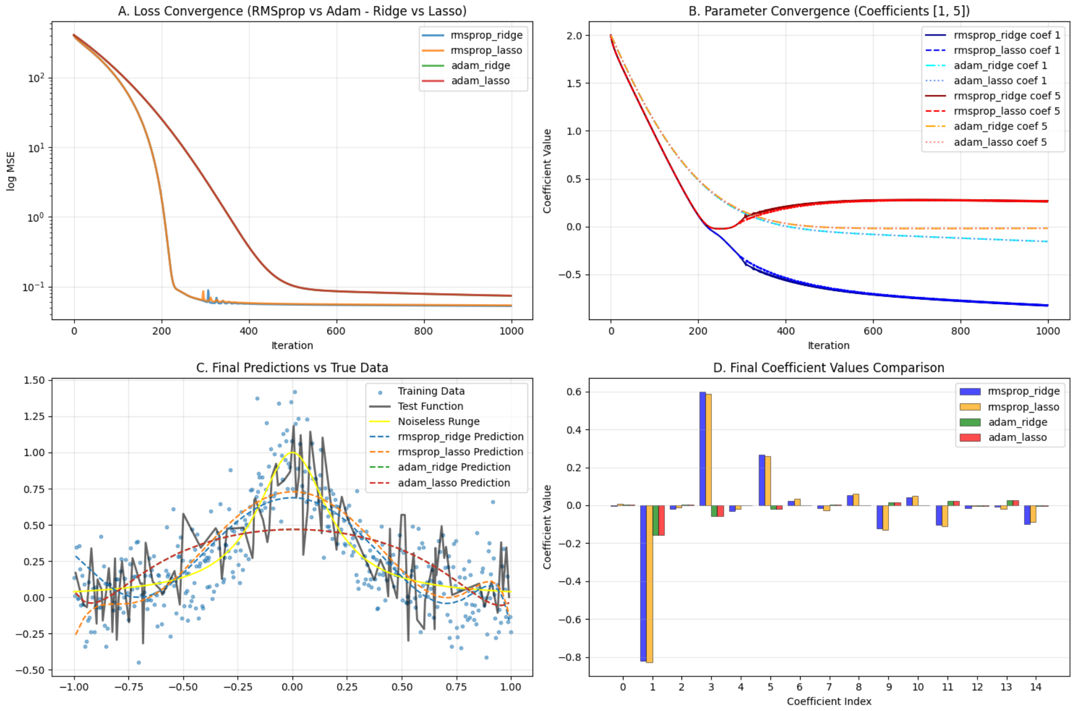

Figure 9: Lasso GD

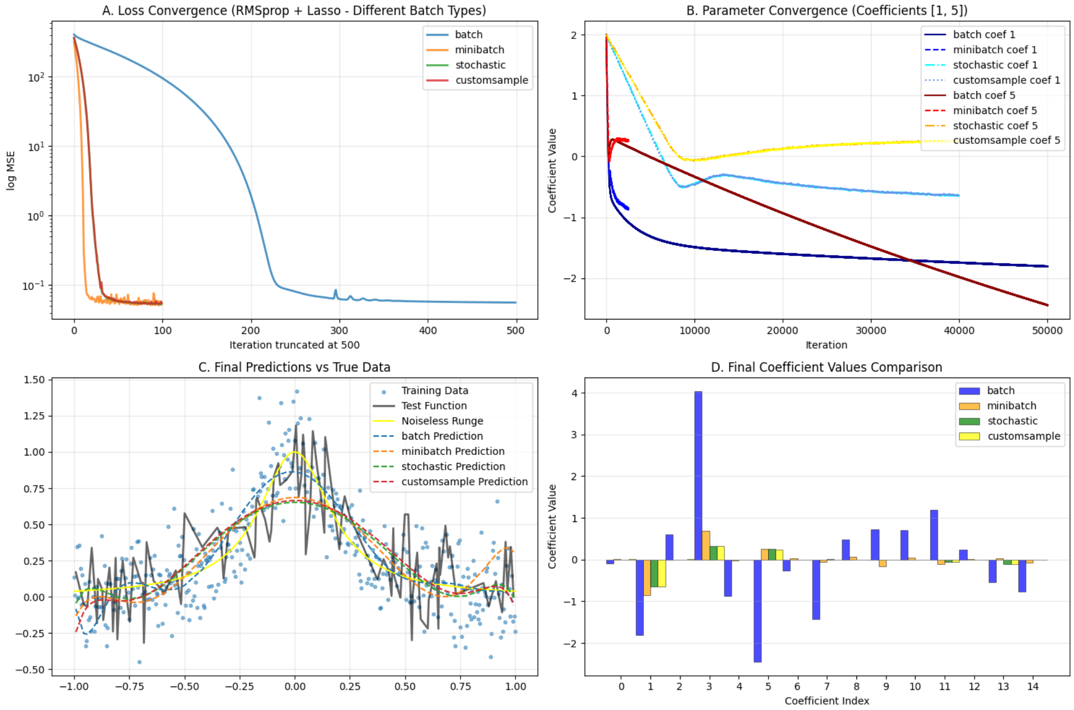

Figure 10: Batching, Minibatching and SGD with Imporance sampling

V. CONCLUSION

Figure 11: My dog. (very cute)

[1] T. Hastie, R.Tibshirani, and J.Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics* (Springer, New York, 2009), URL `https://link.springer.com/book/10.1007%2F978-0-387-84858-7`.

[2] M. Hjorth-Jensen, *Computational Physics Lecture Notes 2015* (Department of Physics, University of Oslo, Norway, 2015), URL `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf`.